A Case for Network-Attached Secure Disks

Garth A. Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang,
Eugene Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri,
Erik Riedel, and David Rochberg

26 September 1996

CMU-CS-96-142

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

garth+nasd@cs.cmu.edu
http://www.cs.cmu.edu/Web/Groups/NASD

Abstract

By providing direct data transfer between storage and client, network-attached storage devices have the potential to improve scalability (by removing the server as a bottleneck) and performance (through network striping and shorter data paths). Realizing the technology's full potential requires careful consideration across a wide range of file system, networking and security issues. To address these issues, this paper presents two new network-attached storage architectures. (1) Networked SCSI disks (NetSCSI) are network-attached storage devices with minimal changes from the familiar SCSI interface (2) Network-attached secure disks (NASD) are drives that support independent client access to drive provided object services. For both architectures, we present a sketch of repartitionings of distributed file system functionality, including a security framework whose strongest levels use tamper-resistant processing in the disks to provide action authorization and data privacy even when the drive is in an physically insecure location.

Using AFS and NFS traces to evaluate each architecture's potential to decrease file server workload, our results suggest that NetSCSI can reduce file server load during a burst of AFS activity by a factor of about 2; for the NASD architecture, server load (during burst activity) can be reduced by a factor of about 4 for AFS and 10 for NFS.

This research is sponsored by DARPA/ITO, through order number D306, and issued by NSWC, Indian Head, under contract N00174-96-0002. Additional support was provided by NSF and ONR graduate fellowships, Hewlett-Packard, Symbios Logic, Data General, EMC, and IBM. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of any sponsoring or supporting agency, including the Defense Advanced Research Projects Agency and the United States Government.

ACM Computing Reviews Keywords: D.4.3 File systems management, D4.6 Access controls C.3.0 Special-purpose and application-based systems.

Introduction

Evolving distributed storage technology for higher performance computing

Distributed file systems provide remote access to common file storage in a networked environment. They enable users of groups of computers to operate as though they were sharing a single large file system [Sandberg85, Howard88, Minshall94].

A principal measure of a distributed file system's cost is the computational power required from the servers to provide adequate performance for each client's work [Howard88, Nelson88]. AFS helps reduce server load by using each client's local disk to cache a subset of the global system's files, allowing a client's local file system cache to handle a large fraction of its distributed file system accesses without contacting the server. This enables AFS servers to support more clients than those distributed file systems, like NFS, whose client caching is limited to the client's in-memory file buffer cache. Of course, in systems that provide strong caching semantics (e.g., AFS), maintaining the consistency of client caches introduces a new, albeit much smaller, computational load on file servers. This load increases as clients cache more aggressively. In addition, there are limits to the effectiveness of client caching — if nothing else, servers must still serve first-reference reads and misses caused by invalidations in client caches.

In large shared file systems, this remaining workload is too large for a single traditional file server. One way to handle the load is to use multiple servers. Multiple-server distributed file systems attempt to balance the load by replicating static, commonly used files and by partitioning the namespace of the remaining files (that is, locating files that are clustered in the same area of the global directory tree on the same server). Namespace balancing is effective when it divides files into sets corresponding to essentially non-overlapping organizational units, but such units are often too large to be serviced by a single low-cost server. Hence, many installations either split the namespace of a single organizational unit over multiple servers or resort to specialized super-fileservers that are large enough to centrally manage all storage for an organizational unit [Hitz90, Drapeau94]. Splitting the namespace leads to the "hotspot" problem familiar from multiple-disk mainframe experience [Kim86], and can require frequent user-directed namespace adjustment. Super-fileservers can provide good performance, but are an expensive solution.

Experience with disk arrays suggests another solution. If the data is striped over multiple independent controllers, then a high-concurrency workload where individual accesses are small relative to the unit of interleaving, will be balanced with high probability [Linvy87, Chen90]. Striping metadata provides similar load-balancing for metadata operations [Dahlin95].

Lowering and balancing the workload applied to servers by each client through client caching and storage striping, respectively, provide excellent cost control, but do not ensure high-performance for clients. With exponential increases in microprocessor performance and the improvements in workstation memory bandwidth required to support them (such as RamBus [Rambus92], or Synchronous DRAM [Toshiba]), ubiquitous personal workstations are increasingly capable of high-performance data processing. Relying on caching to satisfy the

data throughput needs of such high-performance clients would require cache miss rates to decrease proportionately. Unfortunately, increasing computation sizes, file sizes, and workgroup sharing are all blocking the needed decrease in miss rates [Ousterhout85, Baker91], while increased client cache sizes are making those misses more bursty. Thus, if client performance is to improve, the performance of distributed file systems, while servicing client cache misses, must also improve.

This is the argument that led storage subsystem designers to develop disk arrays: striping storage promises parallel transfer of large files and load balancing of high concurrency workloads [Patterson88]. For distributed file systems, striping storage over multiple servers promises scalable storage bandwidth [Hartman93] as long as the network can sustain the communication load. Cost and scalability concerns prohibit the use of a single shared-media network whose peak capacity meets the maximum communication load. However, with the wide acceptance of switched network fabrics based on point-to-point links, such as switched Fast Ethernet, switched FDDI, ATM, and Myrinet, whose links have 100 Mbit/sec to 800 Mbit/sec capacities, striped storage bandwidth can scale up to the limitations of client links, independent of other traffic in the same fabric [Arnould89, Siu95, Boden95]. Of course, a client's network performance is limited by far more than its link's raw bandwidth. Fortunately, there has been substantial research progress toward overcoming many of these performance limitations. Powerful interface board designs [Steenkiste94, Cooper90, FORE94], integrated layer processing for network protocols [Clark89], direct application access to the network interface [vonEiken92, Maeda93], copy avoiding buffering schemes [Druschel93], and routing support for high-performance best-effort traffic [Ma96, Traw95] are all increasing the bandwidth available to client applications.

In practice, distributed file systems are often built as a series of many small purchases made by small groups. Invariably, these small groups are primarily interested in buying client machines. However, the economics of providing a high-performance striped distributed filesystem are like those of purchasing an expensive centralized mainframe: a large investment requiring the financial collaboration of many small groups. One way to avoid synchronizing purchasing and "taxing" the budget of the purchaser of each new machine in the distributed file system domain is to instead tax the machine itself; the xFS filesystem distributes code, metadata and data over all clients, eliminating the need for a centralized storage system [Dahlin95]. This scheme naturally matches increasing client performance with increasing server performance. Instead of reducing the server workload, it takes the required computational power from another, frequently idle, client. The network-attached storage architectures presented in this paper significantly reduce the demand for server computation and eliminate file server machines from the storage data path without coupling overall file system integrity to the security of each client machine.

Storage technology's concurrent evolution

As distributed file system technology has improved, so have the storage technologies employed by these systems. Primarily fixed-surface, flying-head magnetic (hard) disks, a technology developed over three decades ago, storage devices have evolved to provide increasing density, increasing data and seek speeds, and increasing embedded intelligence. Storage density increases, long a predictable 25% per year, have been delivering 60%

increases per year during the 90s. Prior to the mid-80s, data rates were constrained by storage interface definitions, but they have increased by about 40% per year in the 90s [Grochowski96].

The primary reason for storage's recent accelerated rate of evolution has been the broad acceptance of the Small Computer System Interface (SCSI) standard, which abstracted the device as a linear array of fixed-size blocks with an embedded command-interpreting controller and a shared, relatively high-speed bus linking devices to a computer's I/O bus through a "host-bus adapter" [ANSI86]. In contrast to pre-SCSI storage devices, whose interfaces exposed data rates, disk geometry [McKusick84], data-dependent addressing [Ahearn72], and bufferless speed matching (causing so-called rotational positioning reconnect miss delays [Buzen86]), SCSI decouples storage component interfaces from the host's storage interface, enabling rapid introduction of incremental technology advances.

Moreover, the adoption of SCSI (and its less-expensive contemporary, IDE) across a broad range of the marketplace has increased competition among disk drive manufacturers by eliminating the customer's compelling motivation to purchase storage from the vertically integrated provider of the rest of the computer system. The result, 60% per year density increases and 40% per year data rate increases, is now yielding surface densities over 1.3 Gbit per square inch [IBM96], unit capacities over 10 GBytes [Seagate96b] and sustained data rates up to 12 MBytes/sec [Seagate96a]. At this rate of improvement, we can expect data rates in excess of 40 MBytes/sec by the end of the decade.

The level of indirection introduced by SCSI has also led to transparent improvements in storage performance; a device can provide better availability and functionality while exporting the same interface. Notable examples include Redundant Arrays of Inexpensive Disks (RAID); transparent failure recovery; real-time geometry-sensitive scheduling; buffer caching, readahead, and writebehind; compression; dynamic mapping and representation migration [Patterson88, Gibson92, Massiglia94, StorageTek94, Wilkes95, Ruemmler91, Varma95].

Currently, smart storage subsystems contain tens to hundreds of GBytes of storage, service thousands of accesses per second, and easily saturate double and quadruple speed SCSI buses. With this pressure on the performance of SCSI's physical interconnect, the industry is today (1996) experiencing uncertainty about, and rapid development in, peripheral interconnect technologies [Sachs94]. On one hand, traditional SCSI advocates are deploying shorter, faster, wider buses with data rates of 20, 40, and 80 MBytes peak (increasing addressability primarily through hierarchical storage controllers) [ANSI95]. Others, particularly interested in increasing the number and multiplicity of devices and hosts interconnected, have replaced the physical implementation of SCSI with high-speed serial, packetized, ring interconnects such as Fibre Channel (up to 100 MBytes/sec) [Benner96] and SSA (up to 40 MBytes/sec per link[1]) [SSA]. The disk drive industry anticipates the marginal cost for Fibre Channel and SSA interfaces on the disk to be typical of today's Ethernet adapters while their host adapter

1. In SSA, independence of links attached to each node allows multiple point-to-point transfers in parallel where these transfers are physically non-overlapping.

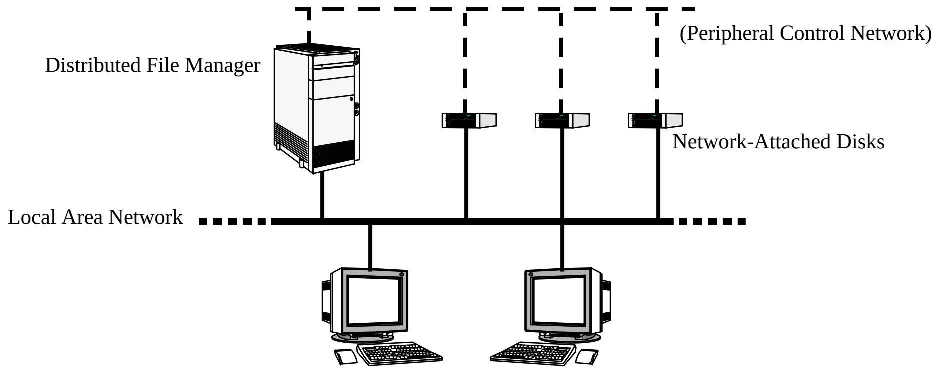

Figure 1: Network-attached storage, in general, provides a direct network connection between client and storage. It may or may not separate higher-level file system function from storage into a file manager machine. It may or may not have a private peripheral network linking file manager and storage devices. Although pictured here as a single-actuator disk drive, a network-attached disk is any device attached to the network and offering storage functionality. That is, for the purposes of this paper, a RAID subsystem can be considered a network-attached disk.

costs are expected to be comparable to high-performance SCSI adapters (between the cost of ethernet and ATM or FDDI interfaces) [Anderson95].

To take advantage of these improvements in network and storage technology, we can attach storage directly to the network. Distributed file systems using network-attached storage scale more cost-effectively for two reasons: server off-loading and network striping. By off-loading simple, expensive, and data-intensive operations from file server machines, more clients and drives can be supported by each file manager machine. By coupling file access computation and network transfer bandwidth to each drive, aggregate transfer bandwidth scales with drives (rather than server memory and network bandwidth) and data avoids a store-and-forward copy through the server machine. In the remainder of this paper, we present an overview of network-attached storage, a taxonomy of network-attached storage, and experiments that attempt to evaluate the performance of the proposed architectures.

Network-Attached Storage

What is Network-Attached Storage?

Figure 1 gives an overview of network-attached storage. This technology, called network-based storage in a trend-predicting paper by Katz [Katz92], is not new.. The Mass Storage System Reference Model (MSSRM), an early architecture for hierarchical storage subsystems, has advocated the separation of control and data paths for almost a decade [Miller88, IEEE94]. Using a high-bandwidth network that supports direct transfers for the data path is a natural consequence [Kronenberg86, Drapeau94, Long94, Lee95, Menascé96]. In the High Performance Storage System (HPSS) [Watson95], the MSSRM model has been implemented and augmented with socket-level striping of file transfers, called the Parallel Transport Protocol [Berdahl95, Wiltzus95], over the multiple network interfaces found on mainframes and supercomputers.

Our notion of network-attached storage is consistent with these projects. However, our analysis focuses on the evolution of commodity storage systems rather than supercomputing systems, and on the interaction of network-attached storage with common distributed file systems. Our goal is to chart the way network-attached storage is likely to appear in products, estimate its scalability implications, and characterize the security and file system design issues in its implementation.

Following Van Meter's [VanMeter96] definition of network-attached peripheral, we consider networks that are shared with general local area network traffic and not single-vendor systems whose backplanes are fast, isolated local area networks [Horst95, IEEE-SCI92].

A taxonomy of network-attached storage architectures

Simply attaching storage to a network underspecifies network-attached storage's role in the distributed file system architecture. In the following subsections we present a taxonomy for the functional composition of network-attached storage.

Case 0, the base case, is the familiar local area network with storage privately connected to the system's file server —we call this *server-attached disks*. Case 1 represents a wide variety of current products, *server-integrated disks*, that specialize hardware and software into an integrated file server product. In Case 2, with current generation disk drives already attaching to peripheral networks, the obvious network-attached disk design, *network SCSI*, minimizes modifications to the drive command interface, hardware and software. Finally, given the rapidly increasing processor capability of the disk-embedded controllers, there is an opportunity to restructure the drive command interface to more effectively off-load data access functionality. In Case 3, we call these higher-function storage devices *network-attached secure disks*.

Case 0: Most storage systems today are Server-Attached Disks (SAD)

This is the system familiar to office and campus local area networks, and is illustrated in Figure 2. Storage is attached privately to general-purpose machines that are dedicated to distributed file service function.

Case 1: Optimized implementations: Server Integrated Disks (SID)

Since file server machines often do little other than service distributed file system requests, it makes sense to construct specialized systems that perform only file system functions and do not perform general-purpose computation. This architecture is not fundamentally different from server-attached disk (hence, it is not separately illustrated). Data must still move through the server machine before it reaches the network, but specialized servers can move this data more efficiently than general-purpose machines. Since high performance distributed file service benefits the productivity of most users, this architecture occupies a high margin (profitable) market niche [Hitz90, Hitz94]. However, this approach binds the client to the chosen distributed file system, its semantics, and its performance characteristics. For example, most server-integrated disks provide NFS file service whose inherent performance has long been criticized [Howard88]. Since the marketplace has not selected a single, high-

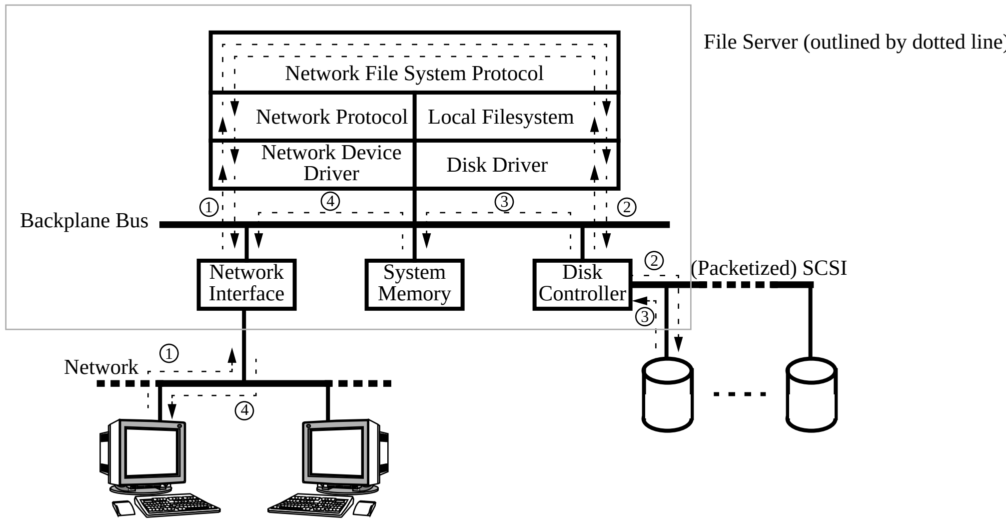

Figure 2: Server-attached disk is the familiar local-area-network distributed file system. A client wanting data from storage sends a message to the file server (1), which parses it and sends a message to storage (2), which accesses the data and sends it back to the file server (3), which finally sends the requested data back to the client (4). Server-integrated disk is logically the same structure, except that the hardware and software in the file server machine is specialized to the file service function.

performance distributed file system, this architecture does not facilitate the development of a SCSI-like commodity market. Also, a critical feature of scalable storage, server striping, is not well-supported by any of the existing popular distributed file systems, so binding the client storage interface to an existing distributed file system is at least premature.

Case 2: Network SCSI (NetSCSI)

An approach at the other extreme from server-integrated disks is to retain as much as possible of the current dominant storage device protocol, SCSI. This is the natural evolution path for storage devices; Seagate's Barracuda FC is already providing packetized SCSI through Fibre Channel network ports to directly attached hosts [Seagate96a]. Network SCSI (NetSCSI), shown in Figure 3, is a network-attached storage architecture that makes minimal changes to the hardware and software of SCSI disks. A file manager machine translates its clients' file system requests into SCSI commands for its disks, but rather than returning data to the file manager to be forwarded, the NetSCSI disk sends data directly to the client. The SCSI COPY command already supports such third-party transfers. By eliminating the file manager from the data's path, its workload per active client decreases. The efficient data transfer engines typical of fast drives ensure that the drive's sustained bandwidth is available to the clients through the network, and that the file manager machine need not be replicated when striping files over many disks for higher bandwidth still. However, the use of third-party transfer changes the drive's role in the overall security of a distributed file system, itself varying from simple accident avoidance in NFS to privacy for all transfers in AFS.

There are four levels of security within the NetSCSI disk model: (1) accident-avoidance with a second private network between file manager and disk, both locked in a physically secure room; (2) data transfer authentication with clients and drives additionally equipped

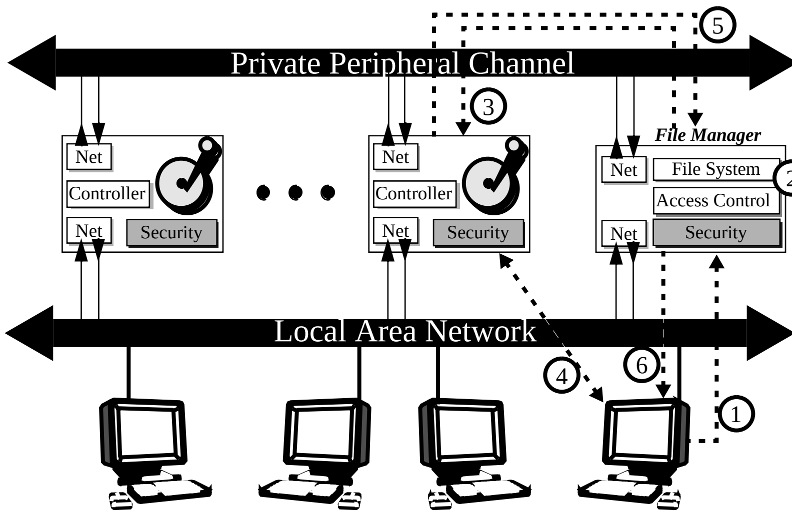

Figure 3: Network SCSI (NetSCSI) is a network-attached disk architecture designed for minimal change in the disk's command interface. However, because the network port on these disks may be connected to the hostile, broader internet, integrity for the file system structure on disk requires a second port to a private, file manager-owned, network. If a client wants data from a NetSCSI disk, it sends a message (1) to the distributed file system's file manager which processes the request normally (2), sending a message over the private network to the NetSCSI disk (3). The disk accesses data, transfers it directly to the client (4), and sends its completion status to the file manager over the private network (5). Finally, the file manager completes the request with a status message to the client (6).

with a strong one-way hash function; (3) data transfer privacy with clients and drives additionally equipped with encryption and; (4) secure key management, where a secure coprocessor removes the need for the disk to be remain physically secure.

Figure 3 shows the weakest NetSCSI security enhancement: a second network port on each disk. Since SCSI disks execute every command they receive without an explicit authorization check, even well-meaning clients can generate arbitrary commands and accidentally damage arbitrary parts of the file structure on disk. The drive's second network port provides accident avoidance while allowing SCSI command interpreters to continue following their execution model; a NetSCSI drive executes all commands arriving over the private network port, rejecting all commands arriving on the general network. This is the architecture employed in the HPSS and SIOF projects at LLNL [Wiltzius95, Watson95]. Assuming that file manager and NetSCSI disks are locked in a secure room, this mechanism is acceptable for the trusted network security model of the NFS distributed file system (which trusts the bits in a packet's header to specify the originating address for authentication) [Sandberg85].

Because file data still travels over the general network which is potentially hostile, NetSCSI disks are likely to demand greater security than the accident avoidance provided by a private network. Cryptographic protocols can strengthen the security of NetSCSI. At a minimum, a strong one-way hash function, such as SHA [NIST94], computed at the drive and at the client may allow data transfer authentication, in that the correct data was received only if the sender and receiver compute the same strong one-way hash on the data. Since error-correcting code hardware is already applied to all data transferred to and from a disk's magnetic media, it should be possible to interpose a strong one-way hash function at the drive without reducing sustained transfer bandwidth.

Data transfer authentication between drive and client is not sufficient to provide transfer privacy. To provide privacy, a NetSCSI disk must be able to encrypt data. With encryption,

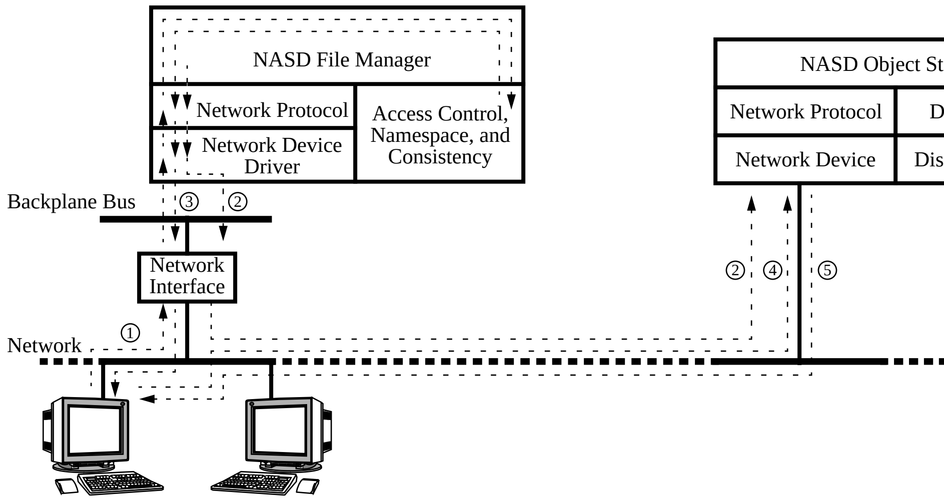

Figure 4: Network-attached secure disks are designed to off-load more of the file system's simple, expensive, and performance critical operations to the storage devices. For example, in one potential protocol a client prior to reading a file, requests access to that file from the file manager (1), which installs into the NASD drive a capability for access to the targeted file (2) and delivers this capability to the authorized client (3). So equipped, this client may make repeated accesses to different regions of the file (4, 5) without further contacting the file manager.

NetSCSI drives can use cryptographic protocols to construct secure virtual channels over the untrusted network. However, since keys will be stored in devices vulnerable to physical attack, the servers must still be stored in physically secure environments, such as a locked room. If we go one step further and equip NetSCSI disks with a secure coprocessor that can securely store keys [Tygar95], data can be stored in encrypted form on the disk, allowing the disks to be used in a variety of physically open environments. There are now a variety of secure coprocessors [Cylink95, NIST94a, Weingart87, White87, Telequip95, National94]available, some of which provide cryptographic accelerators sufficient to support single-disk bandwidths [National96].

Case 3: Network-attached Secure Disks (NASD)

With network-attached secure disks, shown in Figure 4, we relax the goal of minimal change from the existing SCSI interface and implementation. Instead we focus on selecting a command interface that off-loads more of the file manager's work onto the disk without integrating file system policy into the disk. Fast-path operations, like reads and writes, go straight to the disk, and less-common ones, like namespace manipulations, go to the file manager. The disk can present a flat namespace for file-like objects, with pathname resolution split between the file manager and client. While a single drive object will suffice to represent a simple, client file, multiple objects may be logically linked by the file system into one client file. For example, banks of striped files [Hartman93], Macintosh-style forks, file data and metadata, or logically-contiguous chunks of complex files [deJong93].

Clients directly request access to data regions in objects, so a NASD drive must have sufficient metadata on hand to map an object region to a set of magnetic media sectors. This metadata could be provided by the file manager dynamically or it could be maintained by

the drive. While the latter approach asks distributed file system authors to surrender detailed control over layout of the files they create, it enables smart drives to better exploit detailed knowledge of their own resources to optimize data layout, readahead, and cache management [deJonge93, Patterson95]. This is precisely the type of value-added opportunity that nimble storage vendors can exploit for market and, more importantly, customer advantage.

As opposed to NetSCSI, where all drive commands come implicitly authorized from the file manager, NASD drives must authenticate clients and enforce the access control decisions of their file manager on client requests. NASD drives need encryption to provide client authentication, access control enforcement, and data privacy and integrity. If they are further equipped with secure coprocessors, NASD drives can also provide secure key management.

As an example of a possible NASD access sequence, consider a file read (Figure 4). Before a client issues its first read against a file, the client authenticates itself with the file manager and requests access. If the access is granted by the file manager, the client receives the network location(s) of the NASD drive(s) containing the file's objects and time-limited capabilities to present to these drive(s). If the client is new to a drive, it will also obtain a time-limited key for establishing a secure communications channel to the drive. When granting an object capability or communications key to a client, the file manager also informs the corresponding drive using an (independent) channel. After this point the client may directly request access to data on NASD drives, presenting the appropriate capability for each drive to check against the copy provided to it by the file manager.

In addition to off-loading file read operations from the distributed file manager, later sections will show that NASD should off-load to the drive file writes and the reads of file attributes (just another region of a drive's object). Of course, high level file system functions such as access control lists and consistency protocols remain the purview of the file manager which enforces its decisions through its control of the capabilities available in each NASD drive.

Experimental Methodology

To develop an understanding of the performance parameters critical to network-attached storage, we performed a series of measurements to: (1) characterize the behavior and cost of AFS and NFS distributed file server functionality; and, (2) provide data for analytic models of the SAD, NetSCSI and NASD storage architectures.

We began by analyzing AFS and NFS file system traces to determine the types and frequency of operations performed by distributed file systems (Table 1 and Table 2). The NFS traces [Dahlin94] record the activity of an Auspex file server supporting 237 clients over a seven day period at the University of California at Berkeley. These were collected using a packet filter program, rpcspy. The AFS traces record the activity of our laboratory's Sparcstation 20 AFS file server supporting 25 client workstations over a four day period. The AFS traces were collected using a modified version of the AFS logging facility, ViceLog. Both the NFS and AFS traces document every client file system request processed by the file server, recording for each request event an arrival timestamp, a unique client host id, and the type of primitive file system request (e.g., read, write, get/set attributes, open, close).

These traces capture the types and relative frequency of client requests, but they do not include the amount of work done by the file server for each request. To estimate this cost information, we measured NFS and AFS code paths on a current high performance workstation. Specifically, we used Digital Equipment's ATOM binary annotation tool [Srivastava94] to identify the code paths traversed by each type of primitive file system operation on an Alpha workstation, and the Alpha's on-chip cycle counters to measure the amount of work (in CPU cycles) required for each type of primitive operation.

Cost measurements were taken in two steps. For NFS, the entry and exit points of each procedure in the Digital Unix 3.2c kernel were annotated with ATOM to produce a dynamic call graph. The file server machine (a DEC 3000/400 with a 133 MHz Alpha 21064 processor and 64 MB of RAM, running Digital's NFS versthereion 3 server) ran the annotated kernel while NFS client requests were made to the file server, producing a dynamic call graph for each type of primitive NFS operation. We repeated the process for AFS, ATOMizing the AFS user-level server code to produce AFS server call graphs (using a DEC 3000/500 with a 150 MHz Alpha 21064 processor and 128 MB of RAM, running Transarc's AFS version 3.4 server).

After identifying the specific AFS or NFS routines invoked for each type of primitive file system operation, the kernel (and AFS server code) was re-annotate, limiting annotation to the critical components of each primitive operation's code path. This significantly reduced ATOM overhead, minimizing measurement distortion. Primitive file system requests were applied to the selectively annotated kernel (and AFS server) 100 times, generating traces that recorded the code-path execution time. The Alpha's on-chip counters provided single-cycle accuracy for these measurements. For NFS, we calculated and removed the ATOM tracing overhead (although, for AFS, the variability of operation times was too large for calculation and elimination of ATOM overhead). We repeated this process for each operation type to generate the average cost for each primitive file system request, parameterized by request size where appropriate.

Experimental Results

Relative importance of NFS and AFS server operations

Table 1 and Table 2 report the frequency distribution of various server operations for the NFS and AFS traces, respectively. The top of each of these tables lists the operation names, describes their functions, then reports their frequencies and total number of occurrences in the corresponding trace. This data shows that directory read requests (DirReads) are the most frequently executed NFS operations (43%) while attribute read requests (FetchStatus) are the most frequently invoked AFS operation (49%). The results also show that NFS data-moving operations, BlockRead and BlockWrite, account for only 16.9% of all requests. Similarly, AFS's data-moving operations, FetchData and StoreData, account for 23.5% of all AFS requests.

While frequency numbers do not emphasize data-moving operations, the cycle count data shown in the bottom of Table 1 and Table 2 indicate that data movement places a significant

NFS Operation	Description	Percent	Quantity (x10^6)
AttrRead	Get metadata	36.5	11.1
AttrWrite	Update metadata	2.7	0.83
BlockRead	Fetch data from server	14.0	4.25
BlockWrite	Send data to server	2.9	0.88
DirRead	Read directory entries, convert filename to filehandle, etc.	43.1	13.1
DirReadWrite	Creation of files/directories, file renaming, links, etc.	0.7	0.21
DeleteWrite	Deletion of files/directories	0.1	0.04

Size of Operation	Read (x 10^3 cycles)	Write (x 10^3 cycles)
1 byte	54.6	117.5
1K Bytes	61.1	125.1
2K Bytes	68.2	134.5
4K Bytes	78.0	147.8
8,000 Bytes	100.9	199.3

Operation	# of Cycles (x 10^3)
Get Attribute	33
Set Attributes	63
Directory Read (1 entry)	63
Directory Read (40 entries)	105
Directory Lookup	50
Access	37
Remove	135

Table 1: Distribution and cycle costs for each type of NFS operation. All measurements were taken on a DEC 3000/400 (133 MHz) NFS Server running an ATOMized Digital Unix 3.2c kernel. The server's caches were warmed and results from trials that produced misses in the local cache were discarded.

burden on the server. An NFS 8,000 byte read requires over 100K cycles and an NFS write requires almost 200K cycles. AFS incurs more cost for reads and writes: an 8K fetch is 330K cycles and a store is 410K cycles. There are other expensive operations: an NFS directory read (40 entries) requires 105K cycles, and an NFS remove (8K file) consumes 135K cycles, while an AFS BulkStatus (30 entries) requires 1,313K cycles.

To estimate the relative importance of various primitive operations in the total workload applied to a file server, we estimated the total amount of work done per request type by a server during the execution of each trace. Specifically, multiplying the per operation type count of occurrences by the measured average per operation type cycle counts, we estimated the total server workload per operation type. Representing this per operation type total workload as a percentage of the total over all operation types gives our estimate of the relative importance of primitive operations.

As shown in the server-attached disk (SAD) columns of Table 3, the data-moving operations contribute 27% of the total NFS server workload and 59% of the total AFS server workload. This suggests that the performance gained by directly moving data between client and disk may be limited by other file server functionality [Drapeau94]. As the next subsection shows, this observation limits the benefit of NetSCSI for off-loading file manager workload and motivates the design of a NASD drive interface.

Comparing SAD, NetSCSI, and NASD server performance

Based on the analytic model of server workload in SAD systems, described above, and the outline of NetSCSI and NASD drives in the last section, we project the total file manager

AFS Operation	Description	Percent	Quantity (x10^3)
FetchStatus	Query metadata information on a directory or file (creation date, last modified time, permissions, etc.)	49.0	45.7
FetchData	Send data from the AFS server to the requesting client.	18.3	17.1
BulkStatus	Preform a group of FetchStatus operations and package all the results in a single reply	10.6	9.9
StoreStatus	Update metadata information (last modified date, file permissions, etc.)	7.9	7.4
StoreData	Store data sent by a client into a file on the AFS server.	5.3	4.9
CreateFile	Create a new file in the AFS server namespace.	2.1	2.0
Rename	Move a file from one location in the AFS server namepsace to another location.	1.9	1.7
RemoveFile	Delete a file stored on the AFS server.	1.5	1.4
Others	Operations that occurred with a very low frequency (ACL manipulation, symbolic links, directory creation/deletion, lock management, volume management, etc.)	3.4	3.2

	Cycles according to Size of Operation (thousands)										
Operation	0	1	512	1K	2K	4K	8K	16K	32K	64K	1M
StoreData	259	—	291	303	363	371	410	578	750	1,242	16,752
FetchData		179	192	191	204	270	330	439	788	1,544	—
RemoveFile	—	331	396	396	410	411	412	414	429	452	1,053

BulkStatus Size	Cycles (x 10^3 cycles)
1	151
2	154
3	178
10	324
20	578
25	1,313

Operation	Cycles (x 10^3 cycles)
FetchStatus	128
StoreStatus	189
CreateFile	307
Rename	285
Others	227

Table 2: Distribution and cycle costs for each type of AFS operation. Cycle counts were taken on a DEC 3000/500 (150MHz) running an ATOMized AFS version 3.4 server and averaged over 100 trials. The server's caches were warmed and results from trials that produced misses in the local file system cache were discarded. Number of cycles for “Others” (which mainly consists of operations for manipulating callbacks, links, access control lists, and directories) was estimated as the average of the four size-independent operations that were measured individually (namely, FetchStatus, StoreStatus, CreateFile and Rename). Because the variation induced by different levels of instrumentation was insignificant compared to the variation between different trials at the same level of instrumentation, we did not estimate the instrumentation cost.

workload in distributed file systems using network-attached storage. The results of this projection are shown in the NetSCSI and NASD columns of Table 3.

In the NetSCSI model, the read/write data path avoids the file server on data transfers. However, NetSCSI still requires the server to perform processing on every read and write request, specifically to authorize access and determine the block’s location on disk. We modeled the manager workload while employing NetSCSI drives by eliminating the data movement portion of client reads and writes; the work of each write was estimated by the SAD work done by a zero byte store and the work of each read was estimated by the SAD work done by a one byte read.

NFS Operation	SAD		NetSCSI		NASD	
	Cycles ($\wedge 10^9$)	%	Cycles ($\wedge 10^9$)	%*	Cycles ($\wedge 10^9$)	%*
Attr Read	370.6	20%	370.6	20%	0.0	0%
Attr Write	52.3	3%	52.3	3%	52.3	3%
Block Read	367.7	20%	231.9	12%	0.0	0%
Block Write	130.5	7%	103.7	6%	56.0	3%
Dir Read Lookup	254.6	14%	254.6	14%	0	0%
Dir Read non lookup	675.4	36%	675.4	36%	0	0%
Dir RW	13.6	0%	13.6	0%	13.6	0%
Delete Write	5.1	0%	5.1	0%	2.4	0%
Open	0.0	0%	0.0	0%	41.1	2%
Total	**1,869.8**	**100%**	**1,707.2**	**91%**	**165.4**	**9%**

AFS Operation	SAD		NetSCSI		NASD	
	Cycles ($\wedge 10^9$)	%	Cycles ($\wedge 10^9$)	%*	Cycles ($\wedge 10^9$)	%*
FetchStatus	5.8	21%	5.8	21%	0.0	0%
FetchData	10.0	36%	3.1	11%	0.0	0%
BulkStatus	1.7	6%	1.7	6%	0.0	0%
StoreStatus	1.4	5%	1.4	5%	1.4	5%
StoreData	6.5	23%	1.3	5%	0.9	3%
CreateFile	0.6	2%	0.6	2%	0.6	2%
Rename	0.5	2%	0.5	2%	0.5	2%
RemoveFile	0.6	2%	0.6	2%	0.3	1%
Others	0.7	3%	0.7	3%	0.7	3%
Open	0.0	0%	0.0	0%	3.3	12%
Total	**27.8**	**100%**	**15.7**	**56%**	**7.7**	**28%**

Table 3: Projected workload of the NFS (top) and AFS (bottom) distributed file manager. This table reports the estimates of a simple analytic model to compare the relative scalability of file managers in SAD, NetSCSI, and NASD environments. Because NFS and AFS traces are of different servers and lengths, comparison of the two systems is done using the “%” of cycles for each operation, not the total number of cycles for each operation.

* “%” in the NetSCSI and NASD columns represent the percentage difference between each particular NetSCSI or NASD’s operation cycle count and the SAD’s total cycle count.

For AFS, the file manager in a NetSCSI system executes only about half as many cycles as in the SAD system. For NFS, the improvement was much smaller because of NFS’s high frequency of directory and attribute read operations, which are not significantly off-loaded in a NetSCSI system, and because NFS data transfers, typically 8Kbytes per request, are smaller than AFS data transfers, which can be as large as 64Kbytes per request.

In our model of a NASD-based distributed file system all read operations, including attribute and directory reads, are sent directly to the NASD drive by clients. By relying on clients to find attribute and directory data in the NASD object namespace (by convention or

as a result of an open request to the file manager), NASD drives do not distinguish data, attribute and directory operations. For attribute and directory writes, however, we pessimistically assume that clients must send these requests to their file managers. To estimate the manager's pre-authorization and capability setup work, we introduced an open request (synthesized in our traces to occur whenever a file was touched after at least 30 seconds of inactivity) that requires manager work comparable to an attribute write operation. Data write operations, whose data is sent directly to the NASD drive by clients, and file remove operations, whose object deallocation work is done by the NASD drive, are also estimated to require manager work comparable to an attribute write operation. Finally, we assume that NFS clients in NASD systems replace directory lookup operations with NASD (directory) object reads and execute the lookup locally.

For AFS, Table 3 shows that NASD systems may reduce file manager workload by a factor of 2 over NetSCSI systems, or a factor of 4 when compared to SAD systems. For NFS, where directory and attribute reads dominate manager workload, file managers using NASD drives may benefit from a factor of 10 decrease in load.

Conclusion and Future Directions

Network-attached storage, enabling direct transfers between client and storage, can substantially increase distributed file system scalability while simultaneously enabling striped storage to satisfy the bursty, high-bandwidth demands of the increasingly high-performance clients populating local area networks.

In this paper we presented a simple classification of storage architectures for distributed file systems. This classification contains four models. The traditional, server-attached disk (SAD) model is our base case. Server-integrated disk systems include the familiar NFS server products, which are architecturally identical, but with hardware and software designed specifically for executing file service. We do not emphasize this model because it binds storage products to a particular choice of distributed file system.

The remaining two storage models exploit network-attached storage. Network SCSI (NetSCSI) drives are very similar to current SCSI disks in that all file requests go through the distributed file manager, but the resulting data transfers go directly between client and the drive. For AFS workloads, this may reduce file manager workload by about 50%. Different security models can be provided using NetSCSI depending on the cryptographic support provided in the drive. Network-attached secure disks (NASD) support storage semantics between that of block-level protocols like SCSI and distributed file system semantics like NFS or AFS. The partitioning of file system functionality between NASD drive and file manager is optimized to reduce file manager load while maintaining system flexibility. To operate securely in the face of this partition, NASD drives rely on at encryption and key management support. By off-loading data read and write and attribute and directory read operations, distributed file system server load may be reduced by a factor of 4 for NFS to 10 for AFS with NASD drives.

Our analysis is focused on describing the distinct methods of organizing storage architecture and estimating the potential improvement each promises for distributed file systems. With

the promising results given here, our future directions are clear. We plan to demonstrate that distributed file systems can be implemented around network-attached storage, preserving powerful security models, and yielding considerable scalability and client performance advantages. Along this path, many open questions remain. Our NASD model, in particular, expects a disk drive to be capable of computation not normally associated with cost-sensitive commodity peripherals. Drive micro-architectures and software structures must be developed and demonstrated. Server caching in traditional systems is a side-effect of data store-and-forward through the server. With network-attached storage, we lose this benefit, and we must evaluate new caching strategies, including distributing the caches among storage or providing separate cache servers. In the NASD models we have presented, we still assume that clients "open" files by contacting the distributed file system server to set up the state needed for direct transfers to and from storage and allow the file manager to handle consistency. A clear improvement, similar to the effect of client caching in AFS, might be provided by pre-authorization or group-authorization schemes.

Acknowledgments

This research is sponsored by DARPA/ITO, through ARPA Order D306, and issued by Indian Head Division, Naval Surface Warfare Center, under contract N00174-96-0002. Additional support was provided by NSF and ONR graduate fellowships, Hewlett-Packard, Symbios Logic, Data General, EMC, and IBM. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of any sponsoring or supporting agency, including the Defense Advanced Research Projects Agency and the United States Government.

Bibliography

[Ahearn72] Ahearn, G.R., Dichon, Y., and Snively, R.N., "Design Innovations of the IBM 3830 and 2835 Storage Control Units", IBM Journal of Research and Development, Jan. 1972, pp 11-18.

[Anderson95] Anderson, David (Seagate), Personal communication, 1995.

[Anderson96] Anderson, R. and Kuhn, M., "Tamper Resistance - A Cautionary Note", To appear in Proceedings of the 2nd USENIX Workshop on Electronic Commerce, November, 1996.

[ANSI86] ANSI, "Small Computer System Interface (SCSI) Specification", ANSI X3.131-1986, 1986.

[ANSI95] ANSI, "SCSI-3 Fast-20 Parallel Interface", X3T10/1047D Working Group, Revision 6.

[Arnould89] Arnould, E.A. et al., "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers", 3rd ASPLOS, April 1989, pp. 205-216.

[Baker91] Baker, M.G. et al., "Measurements of a Distributed File System", 13th SOSP, Oct. 1991, pp. 198-212.

[Benner96] Benner, A.F., "Fibre Channel: Gigabit Communications and I/O for Computer Networks", McGraw Hill, New York, 1996.

[Berdahl95] Berdahl, L., Draft of "Parallel Transport Protocol Proposal", Lawrence Livermore National Labs, January 3, 1995.

[Boden95] Boden, N.J. et al, "Myrinet: A Gigabit-per-Second Local Area Network", IEEE Micro, Feb. 1995.

[Buzen86] Buzen, J.P. and Shum, A.W.C. "I/O Architecture in MVS/370 and MVS/XA", CMG Transactions, Volume 54, Computer Measurement Group, Chicago IL, Fall 1986.

[Chen90] Chen, P.M. et al., "An evaluation of Redundant Arrays of Disks using an Amdahl 5890," 1990 SIGMETRICS, 1990.

[Clark89] Clark, D.D. et. al., "An Analysis of TCP Processing Overhead," IEEE Communications 27, 6 (June 1989), pp.23-36.

[Cooper90] Cooper, E., et al., "Host Interface Design for ATM LANs", Proceedings of the 16th Conference on Local Computer Networks, Oct. 1991, pp. 247-258.

[Cylink95] Cylink Corp, CY512i press release, February 1995

[Dahlin95] Dahlin, M.D. et al., "A Quantitative Analysis of Cache Policies for Scalable Network File Systems", 15th

SOSP, Dec. 1995.

[deJong93] deJonge, W., Kaashoek, M. F. and Hsieh, W.C., “The Logical Disk: A New Approach to Improving File Systems,” 14th SOSP, Dec. 1993.

[Drapeau94] Drapeau, A.L. et al., “RAID-II: A High-Bandwidth Network File Server”, 21st ISCA, 1994, pp.234-244.

[Druschel93] Druschel, P. and Peterson, L.L., “Fbufs: A High-Bandwidth Cross-Domain Transfer Facility”, 14th SOSP, Dec. 1993, pp. 189-202.

[FORE94] FORE Systems, Inc., “ForeRunner SBA-100/-200 ATM SBus Adapter User’s Manual”, FORE Systems, Inc..

[Gibson92] Gibson, G., “Redundant Disk Arrays: Reliable, Parallel Secondary Storage,” MIT Press, 1992.

[Grochowski96] Grochowski, E.G., Hoyt, R.F., “Future Trends in Hard Disk Drives,” IEEE Transactions on Magnetics 32, 3 (May 1996), pp 1850-1854.

[Hartman93] Hartman, J.H. and Ousterhout, J.K., “The Zebra Striped Network File System”, 14th SOSP, Dec. 1993, pp. 29-43.

[Hitz90] Hitz, D. et al., “Using UNIX as One Component of a Lightweight Distributed Kernel for Multiprocessor File Servers”, Winter 1990 USENIX, pp. 285-295.

[Hitz94] Hitz, D., Lau, J. and Malcolm, M., “File System Design for an NFS File Server Appliance”, Winter 1994 USENIX, Jan. 1994.

[Horst95] Horst, R.W., “TNet: A Reliable System Area Network”, IEEE Micro, February 1995.

[Howard88] Howard, J.H. et al., “Scale and Performance in a Distributed File System”, ACM TOCS 6, 1, Feb. 1988, pp. 51-81.

[IBM96] International Business Machines, Inc., “IBM Travelstar 2XP Datasheet”, 1996.

[IEEE94] IEEE P1244. “Reference Model for Open Storage Systems Interconnection-Mass Storage System Reference Model Version 5”, Sept. 1995.

[IEEE-SCI92] IEEE, “Scalable Coherent Interconnect”, Standard 1596-1992, 1992.

[Katz92] Katz, R. H., “High-Performance Network- and Channel-Attached Storage”, Proceedings of the IEEE 80, 8, Aug. 1992.

[Kim86] Kim, M.Y., “Synchronized disk interleaving”, IEEE Transactions on Computers C-35, 11, Nov. 1986.

[Kronenberg86] Kronenberg, N.P. et al., “VAXClusters: A closely-coupled distributed system”, ACM TOCS 4, 2, May 1986, pp. 130-146.

[Lee95] Lee, E. K., “Highly-Available, Scalable Network Storage”, 1995 Spring COMPCON, March, 1995.

[Livny87] Livny, M., “Multi-disk management algorithms”, 1987 SIGMETRICS, May 1987.

[Long94] Long, D. D. E., Montague, B.R., and Cabrera, L.,“Swift/RAID: A Distributed RAID System,” Computing Systems 7,3, Summer 1994.

[Ma96] Ma, Q., Steenkiste, P., and Zhang, H., “Routing High-Bandwidth Traffic in Max-Min Fair Share Networks,” 1996 SIGCOMM, August 1996.

[Maeda93] Maeda, C., and Bershad, B., “Protocol Service Decomposition for High-Performance Networking”, 14th SOSP, Dec. 1993, pp. 244-255.

[Massiglia94] Massiglia, P., ed., “The RAIDbook”, RAID Advisory Board, 1994.

[McKusick84] McKusick, M.K. et al., “A Fast File System for UNIX”, ACM TOCS 2, 3, Aug. 1984, pp. 181-197.

[Menascé96] Menascé, D.A., Pentakalos, and Yesha, Y., “An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices,” 1996 SIGMETRICS, May 1996.

[Miller88] Miller, S.W., “A Reference Model for Mass Storage Systems”, Advances in Computers 27, 1988, pp. 157-210.

[Minshall94] Minshall, G., Major, D., and Powell, K., “An Overview of the NetWare Operating System”, USENIX Winter Technical Conference, 1994.

[National96] PersonaCard 100 Series. http://www.ipsecure.com/product/pc100ds.htm.

[Nelson88] Nelson, M.N., Welch, B.B. and Ousterhout, J.K., “Caching in the Sprite Network File System”, ACM TOCS 6, 1, Feb. 1988, pp.134-154.

[NIST94] National Institute of Standards and Technology, “Digital Signature Standard.” NIST FIPS Pub 186.

[NIST94a] National Institute of Standards and Technology,”Security Requirements for Cryptographic Modules” NIST FIPS 140-1

[Ousterhout85] Ousterhout, J.K. et al., “A Trace-Driven Analysis of the UNIX 4.2 BSD File System”, 10th SOSP, Dec. 1985.

[Patterson88] Patterson, D.A., Gibson, G. and Katz, R.H., “A Case for Redundant Arrays of Inexpensive Disks (RAID)”, 1988 SIGMOD, June 1988, pp. 109-116.

[Patterson95] Patterson, R.H. et al., “Informed Prefetching and Caching”, 15th SOSP, 1995.

[Rambus92] Rambus Inc., “Rambus Architectural Overview”, 1992. Available at http://www.rambus.com/.

[Ruemmler91] Ruemmler, C. and Wilkes, J., “Disk Shuffling”, Hewlett-Packard Laboratories Concurrent Systems Project Tech Report HPL-CSP-91-30.

[Sachs94] Sachs, M.W., Leff, A., and Sevigny, D., “LAN and I/O Convergence: A Survey of the Issues”, IEEE Computer, Dec. 1994. pp 24-32.

[Sandberg85] Sandberg, R. et al., “Design and Implementation of the Sun Network Filesystem”, Summer 1985 USENIX, June 1985, pp. 119-130.

[Seagate96a] Seagate Corporation, “Barricuda Family Product Brief (ST19171)”, 1996.

[Seagate96b] Seagate Corporation, “Elite Family Product Brief (ST410800)”, 1996.

[Siu95] Siu, K.-Y. and Jain, R. “A brief overview of ATM: Protocol layers, LAN emulation and traffic management”, ACM SIGCOMM, Vol 25, No. 2, December 1994, pp.69-79.

[Srivastava94] Srivastava, A., and Eustace, A., “ATOM: A system for building customized program analysis tools”, WRL Technical Report TN-41, 1994.

[SSA] SSA-User Industry Group “Serial Storage Architecture-SCSI-2 Protocol,” Draft Standard, UIG95SP-9508.

[StorageTek94] Storage Technology Corporation, “Iceberg 9200 Storage System: Introduction”, STK Part Number 307406101, Storage Technology Corporation, Corporate Technical Publications, 2270 South 88th Street, Louisville, CO 80028.

[Steenkiste94] Steenkiste, P. A., “A Systematic Approach to Host Interface Design for High-Speed Networks”, IEEE Computer, March 1994.

[Toshiba96] Toshiba Corporation, “TCS59S1608AFT-10 Data Sheet”, 1996.

[Traw95] Traw, C. B. S. and Smith, J.M., “Striping Within the Network Subsystem”, IEEE Network, July/August 1995.

[Tygar95] Tygar, J.D., and Lee, B.S., “Secure Coprocessors in Electronic Commerce Applications,” Proceedings 1995 USENIX Electronic Commerce Workshop, 1995, New York.

[VanMeter96] Van Meter, R.., “A Brief Survey Of Current Work on Network Attached Peripherals (Extended Abstract)”, Operating Systems Review 30,1, Jan. 1996.

[Varma95] Varma, A. and Q. Jacobson, Q., “Destage Algorithms for Disk Arrays with Non-volatile Caches”, 22nd ISCA, 1995.

[vonEicken92] von Eicken, T. et al., “Active Messages: A Mechanism for Integrated Communication and Computation”, 19th ISCA, May 1992, pp. 256-266.

[Watson95] Watson, R.W., and Coyne, R.A., “The Parallel I/O Architecture of the High-Performance Storage System (HPSS),” 14th IEEE Symposium on Mass Storage Systems, Sept. 1995, pp. 27-44.

[Weingart87] Weingart, S. H., “Physical Security of the μABYSS System”, Proceedings of the IEEE Computer Society Conference on Security and Privacy, 1987, pp 52-58.

[White87] White, S.R. and Comerford, L., “ABYSS: A Trusted Architecutre for Software Protection”, Proceedings of the IEEE Computer Society Conference on Security and Privacy, 1987, pp. 38-51.

[Wilkes95] Wilkes, J. et al., “The HP AutoRAID Hierarchical Storage System”, 15th SOSP, Dec. 1995.

[Wiltzius95] Wiltzius,D. et al., “Network-attached peripherals (NAP) for HPSS/SIOF”, 1995. Available at http://www.llnl.gov/liv_comp/siof/siof_nap.html.